Grundlagen der Informatik III

Wintersemester 2013/2014

Prof. Dr.-Ing. Michael Goesele,
Simon Fuhrmann, Fabian Langguth

TECHNISCHE UNIVERSITÄT DARMSTADT

4. Aufgabenblatt

07.11.2013

Die Aufgaben der Präsenzübung sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind **vom 14.11. bis 21.11.** bei Ihrem jeweiligen Tutor zu Ihrer Präsenzübungszeit in handschriftlicher Form abzugeben. Die Abgaben sollen alle zum Nachvollziehen der Lösungen notwendigen Zwischenschritte, sowie den Namen des Tutors und die Übungsgruppe enthalten. Die Abgaben erfolgen in Teams mit drei Personen, wobei alle Mitglieder des Teams derselben Übungsgruppe angehören müssen. Jedes der Mitglieder muss eine eigene, handschriftliche Lösung abgeben und die anderen Mitglieder sind auf den Lösungen deutlich anzugeben.

Aufgabe 1: Wissensfragen

(a) Listen Sie die drei gebräuchlisten Taxonomien nach Flynn auf und beschreiben Sie kurz deren wesentlichen Merkmale und Unterschiede.

(b) Skizzieren Sie die Konventionen, die bei Unterprogrammaufrufen eingehalten werden sollten. Gehen Sie insbesondere auf die Sicherung von Registerinhalten und Variablenübergabe ein.

(c) Beschreiben Sie kurz den Aufbau der Floating Point Unit (FPU) und erklären Sie die Arbeitsweise ihrer Operationen.

(d) Erklären Sie knapp die Funktionsweise der SSE-Shuffle-Operationen.

(e) Was ist der Unterschied zwischen der Problem-Domäne und der Programm-Domäne?

Aufgabe 2: Floating Point Unit(FPU)

Gegeben ist folgender arithmeteischer Ausdruck $\frac{(b-a)\times(e+f)}{a+(b+c)\times d}$. Sie kennen diesen bereits aus der letzten Hausübung! Setzen Sie für die Variablen a=5.5, b=4.5, c=0.5, d=0.25, e=0.25, f=0.0 und rechnen sie das Ergebnis mithilfe der FPU aus.
Gegeben sei folgendes Codefragment:

```
.data
inout : .string "Ergebnis␣%f\n"
a: .float 5.5
b: .float 4.5
c: .float 0.5
d: .float 0.25
e: .float 0.25
f: .float 0.0

.text

.globl main
main:

#Hier Code einfuegen

subl $8, %esp
fstpl (%esp)
```

```
pushl $inout
call printf

movl $1, $eax
int $0x80
```

Fügen Sie an der angegebenen Stelle FPU-Anweisungen ein, die den obigen arithmetischen Ausdruck berechnen. Nutzen Sie so wenige FPU-Regiser wie möglich.

Aufgabe 3: SSE Shuffling

(a) Gegeben sind die SSE-Register %xmm0 = $|x_3|x_2|x_1|x_0|$ und %xmm1 = $|y_3|y_2|y_1|y_0|$. Geben Sie im Folgenden die Regiserinhalte von %xmm0 und %xmm1 nach den angegebenen Shuffle-Operationen an. Gehen Sie davon aus, dass die Registerinhalte zwischen den einzelnen Aufgaben wieder auf den Anfangszustand zurückgesetzt werden.

(i) `shufps $0xe4, %xmm0, %xmm0`

(ii) `shufps $0xff, %xmm0, %xmm0`

(iii) `shufps $0x39, %xmm0, %xmm0`

(iv) `shufps $0x66, %xmm0, %xmm1`

(v) `shufps $0xd8, %xmm1, %xmm0`

(b) Neben den SSE-Registern aus Aufgabenteil (a) sind nun auch noch die Register %xmm2 = $|a_3|a_2|a_1|a_0|$ und %xmm3 = $|b_3|b_2|b_1|b_0|$ vorhanden. Geben Sie in den folgenden Aufgaben die Shuffle-Befehle an, die den angegebenen Registerinhalt ergeben sowie das SSE-Register, in welchem dieses Ergebnis steht. Gehen Sie davon aus, dass die Registerinhalt zwischen den einzelnen Aufgaben wieder auf den Anfangszustand zurückgesetzt werden.

(i) $|x_3|x_3|a_0|a_0|$

(ii) $|b_0|b_0|a_0|a_0|$

(iii) $|a_1|a_2|a_3|a_0|$

(iv) $|a_2|b_1|b_1|a_2|$

Aufgabe 4: Inline-Assembler-Code in C

Assembler-Programme werden häufig zur Optimierung von Programmteilen eingesetzt, die in einer Hochsprache z. B. C vorliegen. Um den Code nicht vollständig neu zu schreiben, erlauben es moderne Compiler sogenannten Inline-Assembler-Code in ein Programm einzubinden.

Ein einfaches Beispiel zeigt das folgende C Programm:

```
#include <stdio.h>

int main()
{
   int n = 5;

   asm("imul␣%%eax;"
       : "=a" (n)
       : "a" (n)
   );

   printf("Das␣Quadrat␣von␣5␣ist␣%i\n", n);

}
```

Das Programm berechnet das Quadrat der Zahl 5 und gibt das Ergebnis auf dem Bildschirm aus. Inline-Assembler-Code wird mit dem Schlüsselwort **asm** eingebettet.

```
asm ( assembler template
           : output operands (optional)
           : input operands  (optional)
           : list of clobbered registers (optional)
    );
```

a	%eax
b	%ebx
c	%ecx
d	%edx
S	%esi
D	%edi

Die Angabe (Syntax beachten) im obigen Programm stellt also eine Zuweisung der C Variablen n an das Register %eax da. Achtung die Register werden im Inline-Assembler-Code mit %%eax usw. bezeichnet.
Zur Vorbereitung der Hausaufgabe überzeugen Sie sich von der korrekten Ausführung des C Programms mit Inline-Assembler-Code. Modifizieren Sie dann das gegebene Programm, so dass es nicht das Quadrat, sondern die Fakultät einer Zahl berechnet. Alle arithmetischen Operationen sollen dabei als inline Assembler implementiert werden.

Hausaufgabe 1: Assembler, Inline-Assembler in C, SSE (10 Punkte)

Betrachtet wird folgendes C-Programm:

```
#include <stdio.h>

int main()
{

    //Initialisierung
    float A[4][4] = {{1.0, 2.0, 3.0, 4.0},
                     {5.0, 6.0, 7.0, 8.0},
                     {-1.0, 2.0, -3.0, 4.0},
                     {5.0, -6.0, 7.0, -8.0}};
    float B[4][2] = {{1.5, 2.5},
                     {3.5, 4.5},
                     {-5.5, 6.5},
                     {7.5, -8.5}};
    float C[4][2];

    //Berechnung
    float sum;
    for(int i=0; i<4; i++)
    {
        for(int j=0; j<2; j++)
        {
        sum = 0.0;
        for(int k=0; k<4; k++)
            sum += A[i][k] * B[k][j];
        C[i][j] = sum;
        }
    }

    //Ausgabe
    printf("Matrix␣C:␣\n%f,␣%f\n%f,␣%f\n%f,␣%f\n%f,␣%f\n",
        C[0][0], C[0][1], C[1][0], C[1][1], C[2][0], C[2][1], C[3][0], C[3][1]);
}
```

Es führt eine Matrix-Matrix-Multiplikation durch. Das Ergebnis ist eine 4×2-Matrix:

$$\begin{pmatrix} 1.0 & 2.0 & 3.0 & 4.0 \\ 5.0 & 6.0 & 7.0 & 8.0 \\ -1.0 & 2.0 & -3.0 & 4.0 \\ 5.0 & -6.0 & 7.0 & -8.0 \end{pmatrix} \cdot \begin{pmatrix} 1.5 & 2.5 \\ 3.5 & 4.5 \\ -5.5 & 6.5 \\ 7.5 & -8.5 \end{pmatrix} = \begin{pmatrix} 22.0 & -3.0 \\ 50.0 & 17.0 \\ 52.0 & -47.0 \\ -112.0 & 99 \end{pmatrix}$$

(a) Schreiben Sie ein IA32-Assemblerprogramm, das obiges C-Programm realisiert. Testen Sie Ihr Programm auch mit anderen Eingabe-Matrizen. (4 Punkte)
Sie können folgendes Codegerüst verwenden:

```
.data

outstring: .string "Matrix␣C:\n%f,␣%f\n%f,␣%f\n%f,␣%f\n%f,␣%f\n"
matA:   .float 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, -1.0,
                        2.0, -3.0, 4.0, 5.0, -6.0, 7.0, -8.0
matB:   .float 1.5, 2.5, 3.5, 4.5, -5.5, 6.5, 7.5, -8.5
matC:   .float 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0

.text
.globl main
```

```
main:

#TODO: Ihr Code

pushl $outstring
call printf

#TODO: Eventuell Stack aufraeumen

# Exit
movl $0, %eax
ret
```

(b) Ersetzen Sie in dem obigen C-Programm die Berechnung der Matix B durch Ihr IA32-Assemblerprogramm (nur die Berechnung, die Ausgabe kann weiterhin durch den C Code erfolgen). Dazu soll der in dem Präsenzübungsteil vorgestellte Rahmen für Inline-Assembler verwendet werden. Überprüfen Sie, dass das Ihr Programm die gleichen Ausgaben wie das reine C-Programm liefert. (2 Punkte)

(c) Ändern Sie Ihre Inline-Assembler-Implementierung aus b), sodass die Matrix-Matrix-Multiplikation unter Nutzung der SSE-Register parallelisiert wird. (4 Punkte)

Wie immer gilt: **Ihr Code muss auf den Rechnern im RBG-Pool kompilieren und lauffähig sein.**

Sie müssen den Code sowohl bei Ihrem Tutor in der Übung abgeben (gedruckter Code ist erlaubt), als auch im Moodle unter der entsprechenden Abgabe hochladen.

Wir danken Fabian Wagner und Dominik Notz für die Hilfe beim Erstellen der Übung.